From Iteration to System Failure: Characterizing the FITness of Periodic Weakly-Hard Systems

Arpan Gujarati
Max Planck Institute for Software Systems (MPI-SWS), Kaiserslautern, Germany
arpanbg@mpi-sws.org

Mitra Nasri
Delft University of Technology, Delft, Netherlands
m.nasri@tu-delft.nl

Rupak Majumdar
Max Planck Institute for Software Systems (MPI-SWS), Kaiserslautern, Germany
rupak@mpi-sws.org

Björn B. Brandenburg
Max Planck Institute for Software Systems (MPI-SWS), Kaiserslautern, Germany
bbb@mpi-sws.org

Abstract

Estimating metrics such as the Mean Time To Failure (MTTF) or its inverse, the Failures-In-Time (FIT), is a central problem in reliability estimation of safety-critical systems. To this end, prior work in the real-time and embedded systems community has focused on bounding the probability of failures in a single iteration of the control loop, resulting in, for example, the worst-case probability of a message transmission error due to electromagnetic interference, or an upper bound on the probability of a skipped or an incorrect actuation. However, periodic systems, which can be found at the core of most safety-critical real-time systems, are routinely designed to be robust to a single fault or to occasional failures (case in point, control applications are usually robust to a few skipped or misbehaving control loop iterations). Thus, obtaining long-run reliability metrics like MTTF and FIT from single iteration estimates by calculating the time to first fault can be quite pessimistic. Instead, overall system failures for such systems are better characterized using multi-state models such as *weakly-hard constraints*. In this paper, we describe and empirically evaluate three orthogonal approaches, PMC, Mart, and SAp, for the sound estimation of system's MTTF, starting from a periodic stochastic model characterizing the failure in a single iteration of a periodic system, and using weakly-hard constraints as a measure of system robustness. PMC and Mart are exact analyses based on Markov chain analysis and martingale theory, respectively, whereas SAp is a sound approximation based on numerical analysis. We evaluate these techniques empirically in terms of their accuracy and numerical precision, their expressiveness for different definitions of weakly-hard constraints, and their space and time complexities, which affect their scalability and applicability in different regions of the space of weakly-hard constraints.

2012 ACM Subject Classification Computer systems organization → Embedded and cyber-physical systems; Computer systems organization → Real-time systems; Computer systems organization → Reliability; Theory of computation → Probabilistic computation

Keywords and phrases reliability analysis, MTTF/FIT analysis, weakly-hard constraints

1 Introduction

Zero risk of failure in the presence of intolerable errors, such as errors due to electromagnetic interference (EMI), can never be achieved [4]. Instead, since environmental sources of interference are stochastic in nature, probabilistic analyses are often used to bound the

probabilities of failure to within societally acceptable levels of risk. For example, prior work has investigated the effects of EMI on real-time systems (RTS), resulting in the worst-case probability of an EMI-induced message transmission error [15], or an upper bound on the probability of a skipped or an incorrect actuation in a single iteration of a control loop [23].

However, probabilistic analyses for a single message or iteration are often insufficient for answering whole-system reliability questions in the context of system operation times, and for many certification standards, *e.g.*, ISO-26262 for the automotive domain or DO-178C for the avionics domain. For example, given a fleet of one million autonomous cars, each with an average operating time of five hours a day, if it is desired that less than ten cars break down due to EMI in a year, a reliability metric like *Failures-In-Time* (FIT) [46]—*i.e.*, the expected number of failures in one billion operating hours—is much more useful.

A simplistic approach to obtain such long-run reliability metrics from single iteration estimates is to calculate the time to first fault; this can be done analytically if the probability distribution is known, or through simulation otherwise. However, periodic systems, which can be found at the core of most safety-critical RTS, are routinely designed to be robust to a single fault or to a few occasional failures (a classic example being real-time control applications, which are usually robust to a few skipped or misbehaving control loop iterations). Hence, for such well-engineered RTS, this simplistic approach can be excessively pessimistic. Instead, overall system failures of such temporally robust RTS are better characterized using multi-state models such as *weakly-hard constraints*.

Weakly-hard constraints are widely used and well-studied, especially in the context of temporal requirements [11, 12, 16–18, 21, 24, 26, 34, 36, 38]. They capture properties related to a discrete sequence of events (or iterations) rather than properties required per each individual event (or a single iteration). For instance, an (m, k) constraint, one of the simplest forms of weakly-hard constraints, requires a periodic system to have at least m successful iterations in any window of k consecutive iterations. It is non-trivial to obtain closed-form FIT bounds for such stateful specifications. Simulation-based methods do not yield exact answers, may even unsafely under-approximate the true failure rate, and scale poorly, especially when analyzing low-probability events. The reliability modeling literature (see related work in §8) focuses mostly on *spatial redundancy*, *i.e.*, analysis of systems with redundant components, but does not explore analysis of periodic systems with intermittent iteration failures, a form of *temporal redundancy* that is common in RTS, but not in general-purpose systems.

In this paper, we bridge the gap between analyzing the failure probability of a single iteration in a time-sensitive periodic system, *e.g.*, network control systems, and analyzing the overall reliability of the system while considering its robustness to a few failed iterations. That is, we consider the problem of soundly and accurately estimating the FIT rate, or its inverse, the *Mean Time To Failure* (MTTF), of a periodic control system with respect to failure models expressed as one or more weakly-hard constraints, given that bounds on the reliability of a single iteration have been computed, *e.g.*, using the techniques of [15, 23].

Any such analysis to upper-bound the FIT rate (*FITness* analysis) must be *generic* enough to support complex weakly-hard requirements in order to stand for the needs of larger and more complicated systems. Further, a FITness analysis must be *accurate*, ideally, *exact*, to minimize pessimism in the final system reliability. Last, but not least, a FITness analysis must be *scalable* with respect to the problem size, since capturing asymptotic control properties requires dealing with large problem windows. To respond to each of these requirements, we propose and compare three approaches for FITness analysis: PMC, Mart, and SAp.

PMC (**P**robabilistic **M**odel **C**hecking) models the problem as an expected reward problem in a discrete-time Markov chain, which can be solved using state-of-the-art probabilistic model

Table 1 Approaches to FIT derivation.

Approach	Accuracy	Scalability	Expressiveness
PMC	Exact	Poor	General system, all properties
Mart	Exact	Poor	IID systems, all properties
SAp	Approximate	Good	IID systems, single (m, k) constraint

checkers such as PRISM [31] and Storm [19]. PMC is able to express complex robustness constraints as well as sophisticated system models with state-dependent probabilities of failure, such as in [34]. For the common special case of Bernoulli systems, where failure probabilities are independently and identically distributed (IID), martingale theory [32] allows for a direct approach that we call **Mart**. It constructs a system of linear equations, whose solution gives the expected time to failure, and is therefore able to use powerful linear algebra routines such as LAPACK [5] and BLAS [1]. Like PMC, Mart provides an *exact* analysis and can support general weakly-hard constraints, but both PMC and Mart have limited scalability. To scale to large window-size constraints (see §2 for an example), we introduce **SAp** (**S**ound **Ap**proximation), an empirically-driven, scalable, and yet sound approach designed to evaluate a *single* (m, k) constraint. The tradeoffs of the three proposed techniques are summarized in Table 1.

Our main contribution is a *systematic exploration and empirical evaluation* of the aforementioned methods, each of which is sound by construction, for different points in the weakly-hard constraint design space. We show that for (m, k) constraints where m is close to k, exact analyses scale well, but an implementation must account for numerical imprecisions, especially when failure probabilities are low. On the other hand, SAp is scalable across the entire range of m (for a given k) and, in our experiments, provides safe approximations within a factor of two of the exact answer (when both can be computed). While algorithms for computing reliability measures using Markov chains or martingale theory are not new (see, *e.g.*, [39]), to the best of our knowledge, our paper is the first in applying these techniques in the context of weakly-hard periodic RTS, which are at the heart of many safety-critical systems where sound reliability assessments are essential, and in empirically evaluating the performance-accuracy tradeoffs in this context.

The rest of the paper is organized as follows. We start with an example to motivate the reliability analysis problem studied in this paper (§2). A periodic stochastic model of the system and a formal model of weakly-hard constraints are provided in §3. The three FITness analysis approaches, PMC, Mart, and SAp, are discussed in §4–§6, respectively. Results from a comprehensive evaluation of these approaches are discussed in §7. Finally, we conclude with a discussion of related and future work in §8 and §9, respectively.

2 Motivation

As mentioned in §1, this work bridges the gap between single-iteration analyses and full-system reliability analyses for weakly-hard real-time systems. To explain this further, and to motivate our problem statement along with the specific assumptions that we make, we discuss below the steps involved in the end-to-end reliability analysis of a network control system (NCS).

Consider the single-input single-output NCS illustrated in Fig. 1. For mitigating the effects of environmentally-induced transient faults, the NCS consists of active replicas of its sensor and controller tasks. The safety-certification objective is to ensure that despite the presence of transient faults, the system is expected to provide its intended service for at least

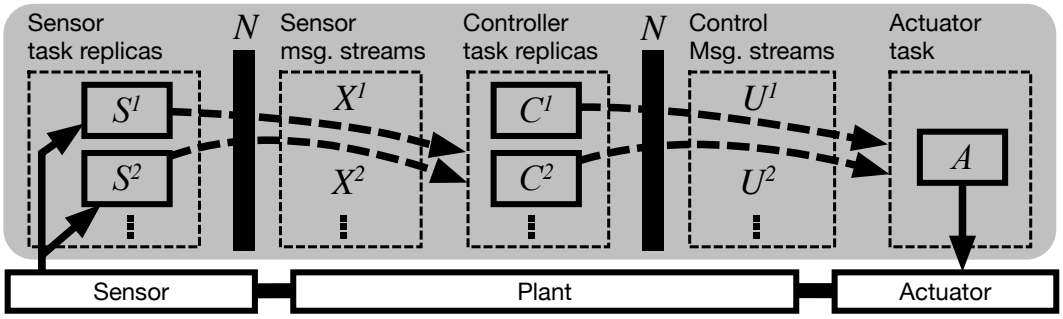


Figure 1 A single-input single-output networked control loop. Solid boxes denote hosts. Each dashed box denotes a task replica set or a set of message streams transmitted by a task replica set. Dashed arrows denote message streams broadcasted over the shared network N.

X hours, where the threshold X is typically determined in a domain-specific manner, and based on whether the system runs in a continuous mode. In other words, the objective is to ensure that the system's MTTF is lower-bounded by X.

The first step in the reliability analysis is to understand the manifestation of transient faults as errors (*i.e.*, program-visible effects of faults). For example, faults on the network can cause retransmission of messages, which may manifest as deadline violation errors if the bandwidth consumed by the retransmissions exceeds the available slack. An upper bound on the probability of such errors can be quantified *a priori* by considering the peak fault rates expected in practice. The next step is to evaluate error propagation in the system. For example, redundant controller tasks with majority voting on the actuator side may mask a few deadline violation errors experienced by the control command messages, but if there are too many such errors, their effect may propagate to the plant actuation stage. Through exhaustive enumeration and evaluation of all error scenarios, the probability of one or more errors affecting the final plant actuation can thus also be upper-bounded. See [15, 20, 23, 25, 43, 44] for such analyses for actual system configurations.

In a nutshell, the above steps provide us with an upper-bound on the per-iteration failure probability, and these bounds are typically independent of the iteration number, since worst-case scenarios and peak fault rates are used in every step of the analysis. Thus, treating an iteration failure as a full-system failure, the per-iteration failure probability bound could be used to estimate the MTTF by calculating the time to first failed iteration. For example, if the NCS loop operates at a frequency of 100 Hz (*i.e.*, with a time period of 10 ms), and its iteration failure probability is upper-bounded by 10^{-10}, its MTTF evaluates to 10^8 seconds (equivalent to a FIT rate of 36 000). However, this simplistic approach can be quite pessimistic, as evident from the estimated FIT rate, which is extremely high. For instance, if the NCS loop functions correctly despite at most one failed iteration in every four consecutive iterations, the estimated FIT rate drops by several orders of magnitude to 1.08×10^{-5}.[1]

In general, prior studies have shown that a control system can be (and typically is) designed to withstand occasionally failing iterations, without compromising its intended service (*i.e.*, the first iteration failure does not denote a full-system failure). For example, Majumdar *et al.* [33] describe a NCS where the control system continues using the previous

[1] The FIT rate of 1.08×10^{-5} for $m = 3$, $k = 4$, $T = 10$ ms, and $P_F = 10^{-10}$ is computed using PMC. In particular, PMC as realized with PRISM yields an MTTF of $T/(3\,600\,000) \times (P_F^3 - 3P_F^2 + 3P_F + 1)/(P_F^4 - 3P_F^3 + 3P_F^2)$ hours. The FIT rate is then computed as FIT $= 10^9/$(MTTF in hours).

iteration parameters in case the current iteration is dropped. Using networked control techniques [14], they also provide methods to estimate a minimum dropout rate tolerated by a control system without compromising its stability (*e.g.*, an inverted pendulum control system with mass 0.5 kg, length 0.20 m, and sampling time 10 ms remains asymptotically stable with at least 76.51 % successful iterations). Such constraints directly translate to weakly-hard models. For instance, the inverted pendulum control system could be safely modeled using $m = 77$ and $k = 100$, or using $m = 766$ and $k = 1000$.

In many cases, however, a single asymptotic constraint of the form $m = 77$ and $k = 100$ may not be sufficient to satisfy other performance specifications (such as settling time), and must be appended with an additional short-range "liveness" constraint. For example, given a sampling time of 10 ms, the inverted pendulum control system would surely crash if it experienced 33 consecutive dropouts. In such cases, the temporal robustness of the control system is better modeled using either a harder constraint (*e.g.*, $m = 4$ and $k = 5$ instead of $m = 77$ and $k = 100$) or multiple constraints (*e.g.*, using both $m_1 = 766$ and $k_1 = 1000$ as well as $m_2 = 1$ and $k_2 = 4$) [13]. The objective of this paper is thus to use the temporal robustness property of control systems, modeled using weakly-hard constraints, for estimating their long-run reliability from the per-iteration failure probabilities.

In the subsequent sections, we provide techniques to estimate the MTTF and FIT of temporally robust periodic RTS with such weakly-hard constraints from their per-iteration failure probability bounds. While these bounds account for the maximum possible background interference, system components that are not being analyzed are assumed to execute reliably. This does not imply that the proposed analysis is not useful if a dependent component fails or if dependent components have different robustness criteria, rather it provides a FIT rate for one subsystem, which can then be composed with the FITs of other dependent, dependee, or unrelated subsystems, *e.g.*, using a fault tree analysis. This is a common way of decomposing the reliability analysis of the whole system into manageable components.

3 System Model

We model the problem of computing a system's MTTF as the expected stopping time of a stochastic process.[2] To that end, we model a periodic system S abstractly as a stochastic process $(X_n)_{n\geq 0}$ evolving in discrete time. We assume that S is periodic with a period of T time units, *i.e.*, the observation X_n is emitted at time nT. Each random variable X_n is boolean-valued: $X_n = 1$ indicates that S executes correctly in its n^{th} period and $X_n = 0$ indicates S executes incorrectly. An *execution* of system S is a string in $\{0, 1\}^*$ denoting an outcome of the stochastic process $(X_n)_{n\geq 0}$. We emphasize that S is *not* just a single, periodic task, but the entire system, divided into logical iterations. For example, as in the system described in §2, one iteration of the system may involve end-to-end execution of a set of periodic real-time tasks and message exchanges (with period T each). The proposed analyses can also be used to analyze multi-rate systems by analyzing each task (or sets of tasks sharing the same rate) individually and adding their respective FIT bounds.

Failure probabilities in system S can be modeled as a *Bernoulli system*, where each observation X_n is an independent, identically distributed (IID) Bernoulli variable, with $Pr[X_i = 0] = P_F$ and $Pr[X_i = 1] = 1 - P_F$. Such a system represents a periodic system

[2] In probability theory [9], a *stochastic process* is defined as a family of random variables R_t, where t ranges over an arbitrary set I. For a given stochastic process, a *stopping time* is a specific type of random variable, whose value is defined as the time at which the stochastic process exhibits a certain behavior of interest (*e.g.*, in this paper, a violation of the system's weakly-hard constraints).

where errors occur independently in each iteration, and the probability of error in each iteration is (bounded by) P_F. It can also represent periodic systems where errors in multiple iterations are dependent, but the bound P_F derived for each iteration is independent of the iteration (this is possible if P_F is derived pessimistically assuming the worst-possible error scenario, which is a common approach in the analysis of hard real-time systems, *e.g.*, [15]).

Alternatively, to capture history-dependence in failures and more accurate iteration-specific error scenarios, the failure probabilities can be modeled more expressively using a *discrete-time labeled Markov chain* [10]. In this case, the system is modeled as a set of states Q and a probabilistic transition function $P : Q \times Q \mapsto [0, 1]$, where $P(s_{n+1}, s_n)$ specifies the probability with which the system transitions from state s_n at any step n to state s_{n+1} at step $n + 1$. Each state is labeled with a Boolean variable denoting success (1) or failure (0), and observation X_n is the label of the (random) state at step n.

Next, we formalize *robustness specifications* to capture the intuition that a periodic RTS, such as one hosting a well-designed control application, continues to provide overall acceptable service despite individual iteration failures, as long as there are not "too many" such iteration failures. In particular, we characterize the set of safe executions for which a periodic system is guaranteed to provide its service as a prefix-closed[3] set of executions $\mathcal{R} \subseteq \{0, 1\}^*$. Thus, the intersection of two robustness specifications is again a robustness specification.

In this paper, we focus on the classic (m, k), $\langle m, k\rangle$, and $\overline{\langle m\rangle}$ robustness specifications, usually called *weakly-hard* specifications, which have been originally proposed in the context of *firm* real-time systems that can tolerate a limited number of deadline misses [12]. Let $\pi_1(s)$ denote the number of 1's (successful iterations) in any string (system execution) $s \in \{0, 1\}^*$. Let $u, v, w, w' \in \{0, 1\}^*$ each denote an execution of system S. Formally, an execution w is (m, k) robust if every window of size k has at least m successes, *i.e.*, $\forall u, v, w' : w = uw'v \wedge |w'| = k \Rightarrow \pi_1(w') \geq m$; it is $\langle m, k\rangle$ robust if every window of size k has at least m consecutive successes, *i.e.*, $\forall u, v, w' : w = uw'v \wedge |w'| = k \Rightarrow \exists u', v' : w' = u'1^m v'$; and it is $\overline{\langle m\rangle}$ robust if there are never more than m consecutive failures, *i.e.*, $\nexists u', v' : w = u'0^{m+1}v'$.

For a given system, one can be interested in several robustness specifications simultaneously, *e.g.*, to express both asymptotic properties (such as "no more than 5% failed iterations") and short-term requirements (such as "no more than two iteration failures in a row"). Thus, for example, we can ask that a system is (m_1, k_1) robust and also $\overline{\langle m_2\rangle}$ robust. This just means that executions of the system satisfy both the (m_1, k_1) constraint and the $\overline{\langle m_2\rangle}$ constraint. In general, given a set of robustness specifications, an execution is considered correct if it satisfies all the specifications in the set.

Given a periodic system S and its robustness specification $\mathcal{R}$, we next define the reliability metrics MTTF and FIT. Let a *system failure* denote an execution that is not in $\mathcal{R}$. For example, for a system with a robustness specification $(2, 5)$, an execution 010100100 denotes a failure (since the last five iterations consist of only one successful iteration). We assume that S stops if it encounters a system failure, and therefore to compute the MTTF and FIT we are interested in a failing execution whose proper prefixes (*i.e.*, prefixes excluding the last iteration) satisfy the robustness specification. Accordingly, given a robustness specification $\mathcal{R}$, we define the *stopping time* of system S as a random variable

$$N(S, \mathcal{R}) = \min \left\{ n \geq 0 \;\middle|\; \begin{array}{l} X_0 \dots X_n \notin \mathcal{R} \wedge \\ \forall i < n \; X_0 \dots X_i \in \mathcal{R} \end{array} \right\}. \tag{1}$$

[3] In a *prefix-closed* set, if an execution belongs to the set, all its prefixes also belong to the set.

The *Mean Time To Failure* (MTTF) is the expectation of the stopping time multiplied by the period T of the system,

$$\text{MTTF} = T \sum_{n=0}^{\infty} n \cdot Pr[N(S, \mathcal{R}) = n]. \tag{2}$$

As mentioned before, the *Failures-In-Time* (FIT) metric is the inverse of the MTTF, with a human-friendly scale factor, to the effect that the FIT represents the expected number of failures in one billion operating hours. Thus, FIT $= 10^9/$(MTTF in hours).

In §4–§6, we propose three approaches for FIT derivation: PMC, Mart, and SAp. To explain the techniques in detail, we initially focus on a single (m, k) robustness specification, and discuss the applicability of the respective technique for evaluating a generic set of robustness specifications such as $\{(m_1, k_1), \ \langle m_2, k_2\rangle, \ \overline{\langle m_3\rangle}\}$ at the end of each section. Wherever a Bernoulli system is considered, P_F is used to denote the probability of a failed iteration, and $P_S = 1 - P_F$ is used to denote the complement of P_F.

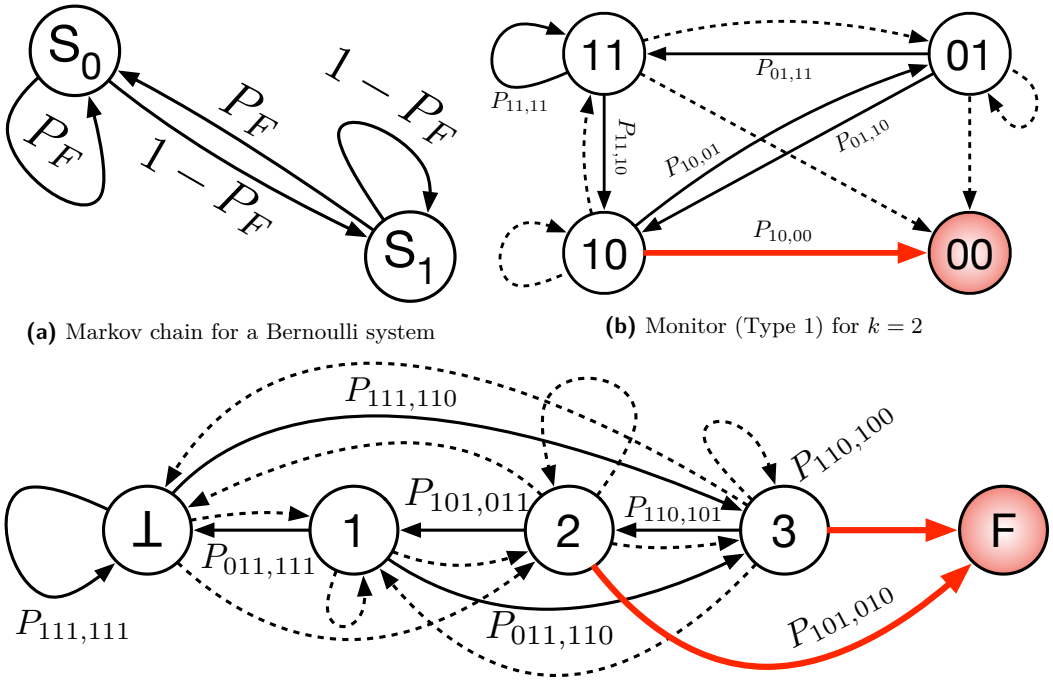
4 PMC: Markov Chain Analysis

We start with the most general method PMC, which is based on discrete-time Markov chains. PMC uses two Markov chains, one for modeling the system, and another (referred to below as the monitor Markov chain) for modeling the weakly-hard robustness constraints as a function of the system's execution history. Therefore, PMC is able to account for both sophisticated system models with state-dependent probabilities as well as complex robustness specifications.

We explain the Markov chain constructions in detail in the following. Our observation is that computing the MTTF reduces to finding the expected total reward in an *absorbing* Markov chain (explained below). Conceptually, our method works for any *regular* robustness specification, *i.e.*, robustness specifications that can be accepted by a finite automaton, but we focus our discussion on the class of weakly-hard robustness specifications, which we expect to be most widely used in practice, and also for concreteness.

Suppose that the system S is modeled as a Markov chain $M = (Q, P, L, s_i)$, where Q denotes a finite set of system states, $P : Q \times Q \mapsto [0, 1]$ denotes the transition probability matrix, $L : Q \mapsto \{0, 1\}$ denotes the state labels with 1 and 0 corresponding to *success* and *failure* (respectively), and $s_i \in Q$ denotes the initial state. For example, if S is a Bernoulli system, then M, as illustrated in Fig. 2(a), consists of states s_0 and s_1 and transition probabilities $P(s_0, s_0) = P(s_1, s_0) = P_F$ and $P(s_0, s_1) = P(s_1, s_1) = 1 - P_F$.

Given the Markov model M and a robustness specification $\mathcal{R} = (m, k)$, we run a *monitor* Markov chain, denoted $\mathit{Monitor}(M, \mathcal{R}) = (Q', P', L', q_i)$, along with M. The monitor tracks a finite execution history of M of length k to decide whether S has *failed*, *i.e.*, whether there were more than $k - m$ failures in the last k steps. Thus, Q' consists of 2^k states, and each state $q \in Q'$ is labeled with a unique label $L'(q) \in \{0, 1\}^k$, *e.g.*, a label of $1^{k-1}0$ implies that every iteration but the last one was successful. Every time M takes a step, the monitor state is updated to reflect the past k steps of M's execution. Thus, the transition probability of $\mathit{Monitor}(M, \mathcal{R})$ from state q with label w to state q' with label w' is $P'(q, q') = P(s, s')$ if system S can transition from history w to w' by transitioning from state s to s'; otherwise, it is $P'(q, q') = 0$. The initial state $q_i \in Q'$ is labeled 1^k to model absence of any failure during system start. In addition, since system S stops as soon as it encounters an execution that does not satisfy (m, k) robustness (recall from §3), we define $\mathit{Bad}(m, k) = \{q \mid q \in Q' \wedge L'(q) \notin \mathcal{R}\}$ as the set of all "bad" states in Q' and make them *absorbing*, *i.e.*, once the monitor enters a state in $\mathit{Bad}(m, k)$, it does not transition into another state.



(a) Markov chain for a Bernoulli system

(b) Monitor (Type 1) for $k = 2$

(c) Monitor (Type 2) for $(2, 3)$

Figure 2 PMC approach. In inset (b), $P_{x_1x_2,\, y_1y_2}$ is a shorthand for transition probability $P'(q, q')$ where states q and q' have labels $L'(q) = x_1x_2$ and $L'(q') = y_1y_2$, respectively. In inset (c), $P_{x_1x_2x_3,\, y_1y_2y_3}$ is a shorthand for $P'(q, q')$, where states q and q' correspond to execution histories $x_1x_2x_3$ and $y_1y_2y_3$, respectively. Since the Type 2 monitor is represented more concisely, the node labels in inset (c) are not equal to the execution histories, *e.g.*, label '3' indicates an execution history of '110' where the latest iteration has failed. In insets (b) and (c), transitions with zero probability are marked with dashed arrows, and states in $Bad(1, 2)$ and $Bad(2, 3)$ are colored red.

As an example, the monitor representation for $\mathcal{R} = (1, 2)$ is illustrated in Fig. 2(b). All execution histories of length $k = 2$ belong to set $\{11, 10, 01, 00\}$, and hence the monitor Markov chain consists of four nodes, each labeled with a unique execution history from this set. An execution history of 01 denotes that the latest iteration was a success (1), whereas the iteration before that was a failure (0). Thus, depending on whether the next iteration is a success or a failure, the system can transition from the state labeled 01 into either a state labeled 11 or 10, respectively. All other transitions from this state have zero probability. Since the robustness constraint $\mathcal{R} = (1, 2)$ defines a robust system execution as one in which at least one out of every two consecutive iterations is successful, the set of bad states in this example is a singleton corresponding to the state labeled 00 with two consecutive failures.

Given the monitor Markov chain construction described above, we reduce the MTTF computation to deriving the expected number of steps until the monitor enters a bad state. For this, assume that each step of the monitor has a reward of 1. We define the *expected number of steps* E as the expected reward until any state in $Bad(m, k)$ is reached, starting from the initial state $q_i \in Q'$. E can be obtained using probabilistic model checkers such as PRISM [31] and Storm [19]. Thus, if system S has period T, and E is the expected number of steps until a state in $Bad(m, k)$ is reached, the MTTF of S with respect to robustness specification (m, k) is given by $\text{MTTF} = T \times E$.

Note that the monitor representation discussed above is independent of m. While the

monitor's simple structure makes it trivial to implement, its $O(2^k)$ space complexity can be detrimental in practice. Fortunately, for the common case where $k - m \ll k$, *e.g.*, $(98, 100)$, the monitor representation can be optimized to be much more space efficient. Since the system stops as soon as the (m, k) constraint is violated, we need not keep any executions that have more than $k - m$ failures. In other words, it suffices to store a limited history as a string of length $k-m$, where each element in the string is from $\{1, \ldots, k\} \cup \{\perp\}$, representing the positions along the previous k steps when a failure occurred ($\perp$ is used in case we have seen fewer than $k - m$ failures). Furthermore, we can coalesce all states in $Bad(m, k)$ into a single "bad" state. The space complexity of the resulting monitor is only $O((k+1)^{(k-m)}+1)$. For example, Fig. 2(c) illustrates the monitor representation for $\mathcal{R} = (2, 3)$, which consists of only five states whereas otherwise it would have required eight states.

Similarly, for $m \ll k$, we can optimize the model by storing a history as a string of length m, where each element in the string is from $\{1, \ldots, k\}$. We refer to the three representations, *i.e.*, the default one, the optimized version for $k - m \ll k$, and the optimized version for $m \ll k$, as Type 1, Type 2, and Type 3 models, respectively.

Compared to the aforementioned monitor representations for an (m, k) robustness specification, monitor representation for $\langle m, k\rangle$ and $\overline{\langle m\rangle}$ robustness specifications are both simpler and more efficient (we do not formally define these due to space constraints). For $\langle m, k\rangle$ robustness, the monitor needs to keep track of positions corresponding to **(i)** the latest run of 1's of length at least m and **(ii)** the current run of 1's of length at most m. For (i), since the beginning and the end of run can be any element in a window of size k, a string of length two belonging to $\{1 \ldots k\}^2$ is needed, whereas for (ii), since the current run must always include the latest element, a string of length one belonging to $\{1 \ldots m\}$ is sufficient. In both cases, $\perp$ can be used to denote the absence of a run, resulting in a space complexity of $O((k + 1)^2 \cdot (m + 1))$. For $\overline{\langle m\rangle}$ robustness, the monitor can be simplified even further, since we only need one accumulator to store the current sequence of consecutive 0's, and so the space complexity is $O(m)$. For multiple specifications, *i.e.*, for robustness constraints of the form $\mathcal{R} = \{(m, k), \langle m', k'\rangle, \overline{\langle m''\rangle}\}$, we run the monitor for each specification in parallel, and set *Bad* to denote states where *some* monitor is in a bad state.

Implementation of the PMC approach using the PRISM probabilistic model checker, along with a discussion of model construction times and model solving times are provided in §7.

5 Mart: The Martingale Approach

Computing the MTTF using PMC reduces to the problem of solving a system of linear equations [10]. In the special case of Bernoulli systems, there is a direct and elegant approach to deriving an equivalent system of linear equations whose solution provides the expected stopping time of the system (*i.e.*, the MTTF), without going through the process of Markov chain modeling. Thus, even though this approach, denoted Mart, does not model history-dependent failures like PMC, it is easy to implement scalably on top of mature linear algebra libraries such as LAPACK [5] and BLAS [1].

We now summarize Mart for (m, k) robustness. The first step in Mart is similar to enumerating the "bad" states of the monitor Markov chain in the PMC approach. In particular, we list all *failure strings* that correspond to a violation of the (m, k) constraint, *i.e.*, all strings in $\{0, 1\}^{\leq k}$ in which at least $k - m + 1$ failures occur. We do this by fixing the last position to be a failure and then choosing all possible combinations of $k - m$ indices from the set $\{1, \ldots, k\}$. In the second step, given an exhaustive list of failure strings, we reduce the problem of computing MTTF to that of computing the *expected waiting time*

until one of the failure strings is realized by the system execution.

To find the expected waiting time, we use an elegant algorithm from the theory of occurrence patterns in repeated experiments proposed by Li [32]. Li's algorithm translates the failure strings into a set of linear equations, such that solving these linear equations directly yields an expected waiting time for each individual failure string (*i.e.*, until a specific failure string is realized by the system) as well as an expected waiting time until any of the failure strings manifests. To compute the MTTF, we require only the latter. We summarize Li's algorithm and MTTF derivation using the algorithm in the following.

Let $\Pi = \{\pi_1, \pi_2, \ldots\}$ be the set of failure strings obtained in the first step. Let $|\pi_i|$ denote the length of a string $\pi_i \in \Pi$, and let $\pi_{i,j}$ denote the j^{th} character in string π_i. Key to Li's algorithm is a combinatorial operator '$*$' (see Eq. 2.3 in [32]) between any pair of strings π_a and π_b from Π, which is defined as follows.

$$\pi_a * \pi_b = (\delta_{1,1}\delta_{2,2}\ldots\delta_{x,x}) \;+\; (\delta_{2,1}\delta_{3,2}\ldots\delta_{x,x-1}) \;+\; \ldots \;+\; (\delta_{x-1,1}\delta_{x,2}) \;+\; (\delta_{x,1}), \tag{3}$$

$$\text{where } x = |\pi_a|,\ y = |\pi_b|,\ \text{and } \delta_{i,j} = \begin{cases} \frac{1}{P_F} & \text{if } i \in [1,x],\ j \in [1,y],\ \pi_{a,i} = \pi_{b,j} = 0 \\ \frac{1}{P_S} & \text{if } i \in [1,x],\ j \in [1,y],\ \pi_{a,i} = \pi_{b,j} = 1 \\ 0 & \text{otherwise.} \end{cases}$$

Using this operator, the expected waiting time e_0 until any one of the sequence patterns in Π occurs for the first time satisfies the following linear system of equations,

$$\begin{bmatrix} 0 & 1 & 1 & \ldots & 1 \\ -1 & \pi_1 * \pi_1 & \pi_2 * \pi_1 & \ldots & \pi_n * \pi_1 \\ -1 & \pi_1 * \pi_2 & \pi_2 * \pi_2 & \ldots & \pi_n * \pi_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ -1 & \pi_1 * \pi_n & \pi_2 * \pi_n & \ldots & \pi_n * \pi_n \end{bmatrix} \begin{bmatrix} e_0 \\ e_1 \\ e_2 \\ \vdots \\ e_n \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \tag{4}$$

where $n = |\Pi|$. Thus, if S has period T, the MTTF is given by $e_o \times T$. As mentioned before, Li's algorithm also yields the expected waiting times for each individual failure string in $\pi_1, \pi_2, \ldots, \pi_n \in \Pi$, which are given by $e_1, e_2, \ldots, e_n$, respectively.

For example, consider a system with period 5 ms, iteration failure probability bounded by $P_F = 0.1$, and robustness specification $(2,3)$, *i.e.*, at most one 0 is allowed in any execution of length three. The set of all failure strings in $\{0,1\}^{\leq 3}$ that violate $(2,3)$ robustness and end in a failure is $\Pi = \{00, 010, 100\}$. Using Eq. 3, $\pi_2 * \pi_2$ is computed as follows.

$$\begin{aligned} \pi_2 * \pi_2 &= \delta_{1,1}\delta_{2,2}\delta_{3,3} + \delta_{2,1}\delta_{3,2} + \delta_{3,1} \\ &= \delta_{1,1}\delta_{2,2}\delta_{3,3} + \delta_{3,1} && \{\text{since } \pi_{2,2} \neq \pi_{2,1},\ \delta_{2,1} = 0\} \\ &= 10 \cdot \delta_{2,2} \cdot 10 + \delta_{3,1} && \{\text{since } \pi_{2,1} = \pi_{2,3} = 0,\ \delta_{1,1} = \delta_{3,3} = 1/P_F = 10\} \\ &= 10 \cdot \delta_{2,2} \cdot 10 + 10 && \{\text{since } \pi_{2,3} = \pi_{2,1} = 0,\ \delta_{3,1} = 1/P_F = 10\} \\ &= 10 \cdot \frac{10}{9} \cdot 10 + 10 = \frac{1090}{9} && \{\text{since } \pi_{2,2} = 1,\ \delta_{2,2} = 1/P_S = 10/9\} \end{aligned}$$

Other $\pi_a * \pi_b$'s can be similarly computed, resulting in the following system of linear equations:

$$\begin{bmatrix} 0 & 1 & 1 & 1 \\ -1 & 110 & 10 & 110 \\ -1 & 10 & 1090/9 & 10 \\ -1 & 0 & 100/9 & 1000/9 \end{bmatrix} \begin{bmatrix} e_0 \\ e_1 \\ e_2 \\ e_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \tag{5}$$

which yields $e_0 = 62.63$ and MTTF $= e_0 \times 5 = 313.15$ ms.

With the MART approach, accounting for a generic set of robustness specifications, such as $\{(m_1, k_1), \langle m_2, k_2\rangle, \overline{\langle m_3\rangle}\}$, is relatively straightforward in comparison to PMC. We need to modify only the first step of MART to obtain an appropriate set of failure strings that corresponds to a violation of any of the robustness specifications, which is used as before to instantiate the system of linear equations defined in Eq. 4. However, we must ensure that any two patterns $\pi_a, \pi_b \in \Pi$ do not contain one another [32]. This is possible if, for example, the failure patterns for constraints $(95, 100)$ and $\overline{\langle 3\rangle}$ are merged. For such cases, the longer pattern is removed from Π, since the shorter pattern occurs first.

6 SAp: An Approximate Analysis

The PMC and MART approaches presented in §4 and §5, respectively, can be used to determine the exact value of MTTF for systems with multiple different types of weakly-hard robustness specifications. Unlike MART, PMC even allows estimating the MTTF of systems that do not resemble a Bernoulli process. However, neither PMC nor MART scale to large values of m and k. Thus, with scalability being the primary motivation, we present next an approximate analysis SAP. Like MART, SAP can be used only for Bernoulli systems. However, unlike PMC and MART, SAP is applicable only for a single (m, k) robustness constraint; it does not support constraints of the form $\langle m, k\rangle$ or $\overline{\langle m\rangle}$, or combinations thereof. Most importantly, though, SAP is *sound*, that is, it estimates an approximate value of the MTTF that lower-bounds the actual MTTF (as given by the exact analyses PMC and MART).

Before we describe SAP, recall the definition of MTTF from §3. For brevity, let $g(n) = Pr[N(S, \mathcal{R}) = n]$, which is a factor in the integrand in Eq. 2. SAP consists of two key steps. In the first step, we derive a lower bound on $g(n)$, denoted $g_{LB}(n)$. For this, we split the (m, k) robustness specification into three conditions, compute an exact or lower bound on the probability for each of these conditions, and then compute a product of these probabilities. In the second step, we integrate $n \cdot g_{LB}(n)$ numerically (but in a sound manner) to strictly lower-bound the MTTF of system S. We discuss both these steps in detail next.

For S to violate the (m, k) specification for the first time during its n^{th} iteration, the following three conditions must hold. $\boldsymbol{(E_1)}$ The n^{th} iteration must fail; $\boldsymbol{(E_2)}$ exactly $k - m$ iterations must fail out of the $k - 1$ iterations between the $(n - k + 1)^{\text{th}}$ and the $(n-1)^{\text{th}}$ iteration; and $\boldsymbol{(E_3)}$ fewer than $k-m+1$ iterations must fail out of any k consecutive iterations, among the first $n - 1$ iterations. Then $g(n) = Pr(E_1) \times Pr(E_2) \times Pr(E_3)$. Now, $Pr(E_1) = P_F$, and summing over all possible combinations of $k-m$ iteration failures in $k-1$ consecutive iterations yields $Pr(E_2) = \binom{k-1}{k-m} P_F^{(k-m)} P_S^{(m-1)}$.

However, obtaining the exact value of $Pr(E_3)$ is challenging. To tackle this challenge, we use the *a-within-consecutive-b-out-of-c:F* model [30, §11.4] (or *a/Con/b/c:F* in short), proposed originally for a system that consists of c ($c \geq a$) linearly ordered components and that fails iff at least a ($a \leq b$) components fail among any b consecutive components. Thus, in terms of the (m, k) constraint, for $a = k - m + 1$, $b = k$, and $c = n - 1$, a successful execution of an a/Con/b/c:F system is equivalent to condition E_3, and the *reliability* of an a/Con/b/c:F system, whose approximations have been well studied in the past, yields $Pr(E_3)$. In particular, since we are interested in a sound approximation, we reuse the reliability lower bound $R_{LB}(a, b, c)$ of the a/Con/b/c:F system as proposed by Sfakianakis, *et al.* [42].

Sfakianakis *et al.*'s analysis [42] breaks the problem into smaller subproblems for which exact analyses are available and that can be computed quickly. However, neither Sfakianakis *et al.* nor any prior work explicitly enumerates the reliability definitions for an exhaustive set of parameters, *i.e.*, which covers all possibles values of parameters a, b, and c. Therefore,

Table 2 Reliability lower bound of a linear a/Con/b/c:F system with IID components. **Type** indicates whether the reliability definition is an exact value or a lower bound (LB).

Case	Definition	Type	Source
$a = 0$	$R_1(a,b,c) = 0$	Exact	–
$a = 1$	$R_2(a,b,c) = P_S^c$	Exact	–
$a = 2,\ c \leq 4b$	$R_3(a,b,c) = \sum_{i=0}^{\lfloor \frac{c+b-1}{b} \rfloor} \binom{c-(i-1)(b-1)}{i} P_F^i P_S^{c-i}$	Exact	[30, §11.4.1] (Eq 11.10)
$a = 2,\ c > 4b$	$R_4(a,b,c) = R_3(a,b,b+t-1)(R_3(a,b,b+3))^u$ where $t = (c-b+1) \bmod 4$ and $u = \left\lfloor \frac{c-b+1}{4} \right\rfloor$	LB	[30, §11.4.1] (Eq. 11.16)
$a > 2,\ c \leq 2b,$ $a = b$	$R_5(a,b,c) = \begin{cases} 1 & 0 \leq c < a \\ 1 - P_F^a - (c-k)P_F^a P_S & a \leq c \leq 2a \end{cases}$	Exact	[30, §9.1.1] (Eq. 9.20)
$a > 2,\ c \leq 2b,$ $a \neq b,\ c \leq b$	$R_6(a,b,c) = \sum_{i=c-a+1}^{c} \binom{c}{i} P_S^i P_F^{c-i}$	Exact	[30, §7.1.1] (Eq. 7.2)
$a > 2,\ c \leq 2b,$ $a \neq b,\ c > b$	$R_7(a,b,c) = \sum_{i=0}^{a-1} \binom{b-s}{i} P_F^i P_S^{b-s-i} M(a',s,2s)$ where $s = c-b$ and $a' = a-i$, and $M(a',s,2s) = \begin{cases} 1 & a' > s \\ R_2(a',s,2s) & a' = 1 \\ R_3(a',s,2s) & a' = 2 \\ R_5(a',s,2s) & a' > 2 \wedge a' = s \\ R_7(a',s,2s) & a' > 2 \wedge a' \neq s \end{cases}$	Exact	[30, §11.4.1] (Eq. 11.14)
$a > 2,\ c > 2b$	$R_8(a,b,c) = R_\phi(a,b,b+t-1)(R_\phi(a,b,b+3))^u$ where $t = (c-b+1) \bmod 4$ and $u = \left\lfloor \frac{c-b+1}{4} \right\rfloor$, and $R_\phi(a,b,c) = \begin{cases} R_5(a,b,c) & a = b \\ R_6(a,b,c) & a \neq b \wedge a \leq b \\ R_7(a,b,c) & a \neq b \wedge a > b \end{cases}$	LB	[30, §11.4.1] (Eq. 11.16)

we provide an unambiguous definition of the reliability lower bound $R_{LB}(a,b,c)$ that draws from Sfakianakis *et al.*'s analysis for large values of c and from other prior works for some special cases and smaller values of c. Note that in many cases, there are multiple ways to define $R_{LB}(a,b,c)$, in which case we prefer a definition that can be quickly computed. We summarize our definition of $R_{LB}(a,b,c)$ in Table 2.[4] Using this reliability lower bound and the definitions of $Pr(E_1)$ and $Pr(E_2)$, a lower bound $g_{LB}(n)$ on $g(n)$ is given by

$$g_{LB}(n) = \binom{k-1}{k-m} P_F^{(k-m+1)} P_S^{(m-1)} R_{LB}\left(k-m+1, k, n-1\right). \tag{6}$$

The next step is to use $g_{LB}(n)$ for lower-bounding the system's MTTF. This requires solving Eq. 2 with $g_{LB}(n)$ in place of $Pr[N(S,\mathcal{R}) = n]$. Unfortunately, we were not able to obtain a closed-form solution with current symbolic solvers due to the complicated definition of $g_{LB}(n)$. In particular, $g_{LB}(n)$ is defined in terms of $R_{LB}(k-m+1, k, n-1)$, which

[4] In our definition of $R_{LB}(a,b,c)$ in Table 2, notice that while we are interested in a reliability lower bound, we point to Eq. 11.16 in [30, §11.4.1], which refers to an upper bound. This mismatch is due to a slight inconsistency in how the textbook chapter [30, §11.4.1] adopts the result from the original paper by Sfakianakis *et al.* [42]. Notations L and U in Table I in [42] denote lower and upper bounds (respectively) on the failure rate of the system. Eq. 11.16 in [30, §11.4.1] uses the same notation. Thus, UB_a in Eq. 11.16 in [30, §11.4.1] actually refers to an upper bound on the system failure probability, and not an upper bound on the system reliability (although the text in the chapter may seem contradictory). Since we require a lower bound on the system reliability, and since system reliability is one minus its failure probability, we use $1 - UB_a$, where UB_a is defined as in Eq. 11.16 in [30, §11.4.1].

is a recursive expression with complex definitions of its subproblems, as can be seen from Table 2. Therefore, similar to numerical integration methods, we adopt an empirical solution for MTTF derivation that is both fast and reasonably accurate. We empirically compute the value of function $g_{LB}(n)$ at finitely many sampling points $d_0, d_1, d_2, \ldots, d_D \in \mathbb{N}$ such that $d_0 = k - m + 1$, and $d_0 < d_1 < d_2 < \ldots < d_D$. Using the empirically-determined values $g_{LB}(d_0)$, $g_{LB}(d_1)$, $\ldots$, $g_{LB}(d_D)$, we derive a lower bound on the MTTF in Lemma 1 below.

The derivation in Lemma 1 depends on the property that $g_{LB}(n)$ (defined in Eq. 6) decreases with increasing n, which in turn requires that $R_{LB}\,(a,\, b,\, c)$ decreases with increasing c (since all the terms except $R_{LB}\,(k-m+1,\, k,\, n-1)$ in the definition of $g_{LB}(n)$ are independent of n). While this property trivially holds for cases $a = 0$ and $a = 1$, proving the property for cases $a = 2$ and $a > 2$ is not trivial. We have provided a detailed proof of this monotonicity property for the more general case $a \geq 2$ online [22, Section IV.B].

▶ **Lemma 1.** *A lower bound on the MTTF of system S with period T and robustness specification $\mathcal{R} = (m, k)$ is given by $T \sum_{i=0}^{D-1}(g_{LB}(d_{i+1}) \times (d_{i+1} - d_i) \times (d_i))$.*

Proof. From Eq. 2, MTTF is defined as $T \sum_{n=0}^{\infty} n \cdot Pr[N(S, \mathcal{R}) = n]$. Since $g_{LB} \leq g(n) = Pr[N(\mathcal{S}, \mathcal{R}) = n]$, we lower-bound the MTTF as $MTTF \geq T \sum_{n=0}^{\infty}(n \times g_{LB}(n))$.

Next, we split the summation range $(0, \infty)$ in the above equation into a finite number of subintervals $(0, d_0]$, $(d_0,\, d_1]$, $\ldots$, $(d_{D-1},\, d_D]$, and $(d_D,\, \infty)$. Further, since all terms under the summation are non-negative, and since we are interested in a lower bound, we drop the summation terms corresponding to subintervals $(0, d_0]$ and $(d_D,\, \infty)$. Thus, we obtain another lower bound MTTF $\geq T \sum_{i=0}^{D-1} \sum_{n=d_i}^{d_{i+1}}(n \times g_{LB}(n))$.

Now, since $g_{LB}(n)$ is decreasing with increasing n, for each interval $(d_i,\, d_{i+1}]$, we replace $g_{LB}(n)$ with $g_{LB}(d_{i+1})$, which is a constant with respect to n. With this replacement, we get MTTF $\geq T \sum_{i=0}^{D-1}(g_{LB}(d_{i+1}) \times \sum_{n=d_i}^{d_{i+1}} n)$. Finally, summing the arithmetic progression, and using inequalities $d_{i+1} - d_i + 1 > d_{i+1} - d_i$ and $d_i + d_{i+1} > 2d_i$, we get the desired bound:

$$\begin{aligned}\text{MTTF} &\geq T \sum_{i=0}^{D-1} \left(g_{LB}(d_{i+1}) \times \sum_{n=d_i}^{d_{i+1}} n \right) \\ &\geq T \sum_{i=0}^{D-1} \left(g_{LB}(d_{i+1}) \times (d_{i+1} - d_i) \times (d_i) \right).\end{aligned}$$

◀

Since scalability is the primary motivation for SAp, we choose $D \ll d_D$, so that the MTTF lower bound can be quickly computed using Lemma 1. We further choose the sampling points $d_1, \ldots, d_D$ to minimize the amount of pessimism introduced by numerical integration. Another source of inaccuracy is the use of the reliability lower bound $R_{LB}(a, b, c)$ proposed by Sfakianakis *et al.* [42], which inherently introduces some pessimism. We discuss the choice of sampling points in detail in §7, and compare SAp with PMC and Mart in terms of accuracy.

As mentioned before, SAp is customized for a single (m, k) constraint and does not apply to $\langle m, k \rangle$ or $\overline{\langle m \rangle}$ robustness specifications. We leave similar approximate analysis for the other robustness constraints as future work.

7 Evaluation

The objective of this section is threefold. We discuss implementation choices and challenges, compare the three types of Markov chain models discussed in §4, and then explore the scalability versus accuracy tradeoffs of PMC, Mart, and SAp. Since the approximate analysis SAp is not applicable to generic robustness specifications as defined in §3, and since

Table 3 MTTF values derived using PRISM engines

Engine	Iterations	Epsilon	MTTF for $P_F = 10^{-2}$	MTTF for $P_F = 10^{-10}$
Explicit	10^{04}	10^{-06}	–	–
	10^{09}	10^{-06}	3.36×10^{05}	0.23×10^{15}
	10^{09}	10^{-10}	3.41×10^{05}	1.21×10^{17}
Exact	N/A	N/A	3.41×10^{05}	3.33×10^{29}

(m, k) constraints are the limiting factor when it comes to scaling up the analysis, we focus on Bernoulli systems and a single (m, k) constraint in the evaluation. In the end, we revisit the strengths and weaknesses of each approach. All experiments were carried out on Intel Xeon E7-8857 v2 machines with 4×12 cores and 1.5 TB of memory.

7.1 Implementation Choices

In the following, we highlight important implementation choices that affect the accuracy and speed of the analyses. We realized PMC using the state-of-the-art probabilistic model checker PRISM [31].[5] However, configuring PRISM properly to ensure that the estimated results are both accurate and sound is not trivial. PRISM provides many different configuration options that affect the method used for linear equation solving (*e.g.*, Jacobi, Gauss-Seidel, *etc.*), the model checking engine (MTBDD, Sparse, Hybrid, or Explicit), parameters for precision tuning (*i.e.*, the *epsilon* value and maximum number of iterations for convergence checks during iterative linear solving), and even options to select *exact* (with arbitrary precision) or *parametric* model checking (where some model parameters are not fixed). Choosing the right set of options is thus important because they can significantly affect the estimated MTTF.

With the parametric model checking option, PRISM outputs the MTTF as a function of parameter P_F, *e.g.*, denoting P_F as q, the MTTF for robustness specification $(2, 4)$ is:

$$T \times \frac{q^5 - 3q^4 + 3q^3 - 2q^2 - q - 1}{q^6 - 3q^5 + 4q^4 - 3q^3}. \tag{7}$$

Parametric model checking is thus an ideal choice since it allows for fast reliability analysis across a range of failure probabilities without the need to build and check the model repeatedly. However, as we show later, parametric model checking is also the costliest analysis approach. Thus, for scalability purposes, we also considered both exact and non-exact model checking.

We observed that non-exact model checking resulted in significant inaccuracy. For example, Table 3 reports the MTTF results for specification $(2, 4)$ obtained with non-exact model checking (using PRISM's Explicit engine) and with exact model checking (currently implemented by PRISM as a special case of parametric model checking). The non-exact engine did not converge (first row of the table) for default configuration options. For $P_F = 10^{-10}$, even upon decreasing the epsilon value and increasing the maximum number of iterations, the estimated MTTF is several orders of magnitude off from the exact value, indicating the sensitivity of non-exact model checking to small probabilities. In our evaluation of PMC, we thus worked only with parametric and exact model checking. We denote these variants of PMC as PMC-P and PMC-E, respectively.

[5] Our implementation of PMC using PRISM for a robustness specification of $(5, 10)$ is explained in the appendix. An empirical comparison of PRISM with Storm [19], a more recent probabilistic model checker, is also provided in the appendix.

Table 4 Percentage errors in FIT ($\mathcal{R} = (8, 10)$ and $y = 1.23456789$)

Precision	$P_F = y \cdot 10^{-10}$	$P_F = y \cdot 10^{-30}$	$P_F = y \cdot 10^{-50}$
10	-2.20×10^{-00}	-3.96×10^{-01}	-1.42×10^{-00}
20	$+1.81 \times 10^{-04}$	-2.70×10^{-04}	$+3.04 \times 10^{-04}$
30	$+3.39 \times 10^{-07}$	-5.26×10^{-07}	$+1.36 \times 10^{-06}$
40	-2.75×10^{-10}	$+1.20 \times 10^{-09}$	-2.00×10^{-09}
50	-1.89×10^{-14}	$+2.99 \times 10^{-13}$	-4.80×10^{-13}

The MART approach was implemented in C++ using the *Elemental* [2] library, since it uses LAPACK-based routines [5] for solving linear equations, allows for arbitrary precision using the GNU MPFR library [3], and also allows for parallel computing using OpenMPI [7]. SAP was implemented in Python using the *mpmath* [6] library for arbitrary precision. Thus, for MART and SAP, unlike for PMC-E, we could explicitly set the global working *precision*, *i.e.*, the number of decimal digits used to represent the floating point significand.

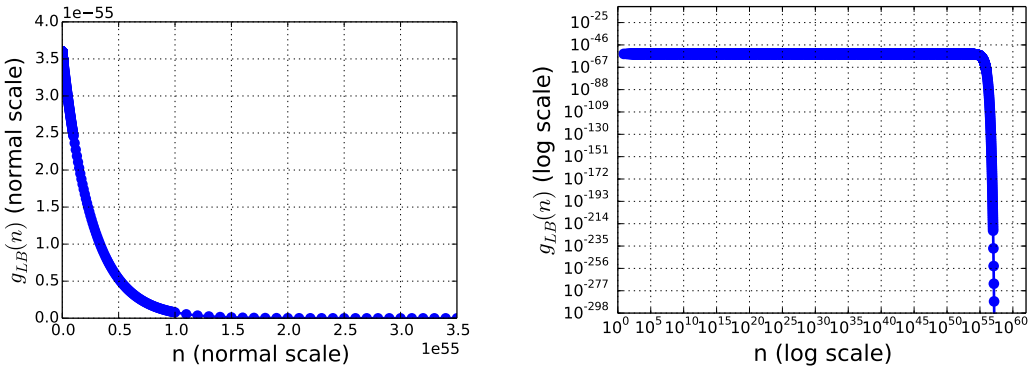
However, the choice of the global working precision was not obvious. Table 4 reports the percentage errors in the estimated FIT when the precision is varied from 10 to 50, with respect to the FIT estimated using a precision of 1000. The results indicate that low precision may result in significant errors if P_F is also small, and sometimes, the results can even be unsafe (*i.e.*, resulting in negative errors). In general, estimating a precision that is safe to use based on the computations involved requires rigorous analysis, *e.g.*, [28]. To be on the safe side, we used a precision of 1000 bits for both MART and SAP, which ensured that any remaining errors were of negligible magnitude.

Finally, when implementing SAP, recall that we need a mechanism to choose an appropriate set of data points d_0, d_1, d_2, ..., d_D over which to run the empirical computations. We discuss this mechanism with the help of an example. Let $m = 3$, $k = 10$, and $P_F = 10^{-7}$. In Fig. 3(a), we illustrate $g_{LB}(n)$ given these parameters. Since the MTTF lower bound derived using SAP depends on $g_{LB}(n)$, the key idea is to ensure that points d_0, d_1, d_2, ..., d_D are sufficient to trace the shape of function $g_{LB}(n)$, and that the magnitude of $g_{LB}(n)$ is negligible beyond $n = d_D$. The first point d_0, as mentioned before, is set to $(k - m + 1)$. To compute the last point d_D, *i.e.*, the point at which $g_{LB}(n)$ becomes negligible, we observed the logarithm of function $g_{LB}(n)$ for $n \in \{1, 10^1, 10^2, 10^3, \ldots\}$. That is, we plotted the function $g_{LB}(n)$ on a logarithmic scale for both the x- and y-axes as in Fig. 3(b), and then determined a threshold at which the curve starts falling rapidly (*e.g.*, $d_D \approx 10^{55}$ in Fig. 3(b)).

The intermediate points $d_1, d_2, \ldots, d_{D-1}$ were chosen such that the step size $d_{i+1} - d_i$ between any two consecutive points d_i and d_{i+1} **(i)** is small enough to closely track the function $g_{LB}(n)$, and **(ii)** yet still proportional to the order of magnitude of d_i, to avoid evaluating an exponential number of points. For example, while generating Fig. 3, the step size was 1 for $n \in (10, 100]$ and 10^{52} for $n \in (10^{53}, 10^{54}]$.

7.2 Evaluating PMC Model Types

Recall from §4 that we introduced three different types of Markov chain models—Type 1, Type 2, and Type 3—each resulting in a different asymptotic model size. Does the use of one model over the other affect the computation times or even the model building times in practice? To answer this question, we measured the asymptotic model sizes for $k = 20$ and $m \in [1, k - 1]$, and compared the measurements with the model size and build time statistics



(a) Normal-scale axes **(b)** Log-scale axes

Figure 3 Sampling points $g_{LB}(d_0)$, $g_{LB}(d_1)$, ..., $g_{LB}(d_D)$ for $m = 3$, $k = 10$, $P_F = 10^{-7}$, and $T = 10\,\text{ms}$ in (a) normal scale and (b) log scale. In this example, $D = 5050$ and $d_D = 9.90 \times 10^{57}$.

reported by PRISM. We also measured the checking time statistics for $k = 10$ (since model checking for $k = 20$ frequently timed out). We summarize the results for PMC-E in Fig. 4.

Fig. 4a plots the asymptotic size for each model type, indicating that none of the models is an optimal choice for all parameters. Fig. 4b reports the number of elements in the transition matrix as reported by PRISM. The number of transition matrix nodes varies with m in the same way as the asymptotic model size, but the absolute numbers are less than the asymptotic sizes. This is because PRISM already prunes some states that are unreachable during the build process. Figs. 4c and 4d illustrate the time to build and check the models, respectively. The model construction time for each model type is proportional to the respective model size. The model checking time, however, is independent of the model type, since the models are equivalent and result in the same set of linear equations.

In summary, to achieve maximum scalability, it is important to choose a model that requires the minimum time for construction. In the subsequent experiments, we thus use the asymptotic model sizes as a guideline to choose the appropriate model type for an (m, k) specification. That is, if $k = 20$, based on Fig. 4a we use the Type-3 model if $m \leq 4$, the Type-2 model if $m \geq 16$, or the Type-1 model otherwise.

7.3 The Scalability vs. Accuracy Tradeoff

We start by evaluating the scalability of the analyses PMC-P, PMC-E, Mart, and SAp by measuring the analysis duration for each $k \in [2, 20]$ for four different configurations of m and P_F: **(i)** $m = \lfloor k/2 \rfloor$ and $P_F = 10^{-10}$ (Fig. 5a); **(ii)** $m = \lfloor k/2 \rfloor$ and $P_F = 10^{-20}$ (Fig. 5b); **(iii)** $m = k - 2$ and $P_F = 10^{-10}$ (Fig. 5c); and **(iv)** $m = k - 2$ and $P_F = 10^{-20}$ (Fig. 5d). Since evaluating (m, k) requires maximum time if $m = k/2$ and minimum time if m is close to either 1 or $k - 1$ (see Fig. 4d), results for (i) and (iii) indicate the minimum and maximum scalability that can be achieved by the analyses; whereas results for (ii) and (iv) help us to understand the impact, if any, of P_F's value on the analysis scalability.

First, as evident from each graph, and as expected, PMC-P, PMC-E, and Mart do not scale well in comparison to SAp. For configurations of type (i) and (ii), where $m = \lfloor k/2 \rfloor$ (see Figs. 5a and 5b), PMC-P and PMC-E scale only up to $k = 9$ and $k = 11$, respectively. The Mart approach performs better and scales up to $k = 15$, mainly because it gives up exactness (but still guarantees soundness owing to its very high precision). In contrast, SAp easily scales up to the maximum value of $k = 20$. Also, notice that while SAp's

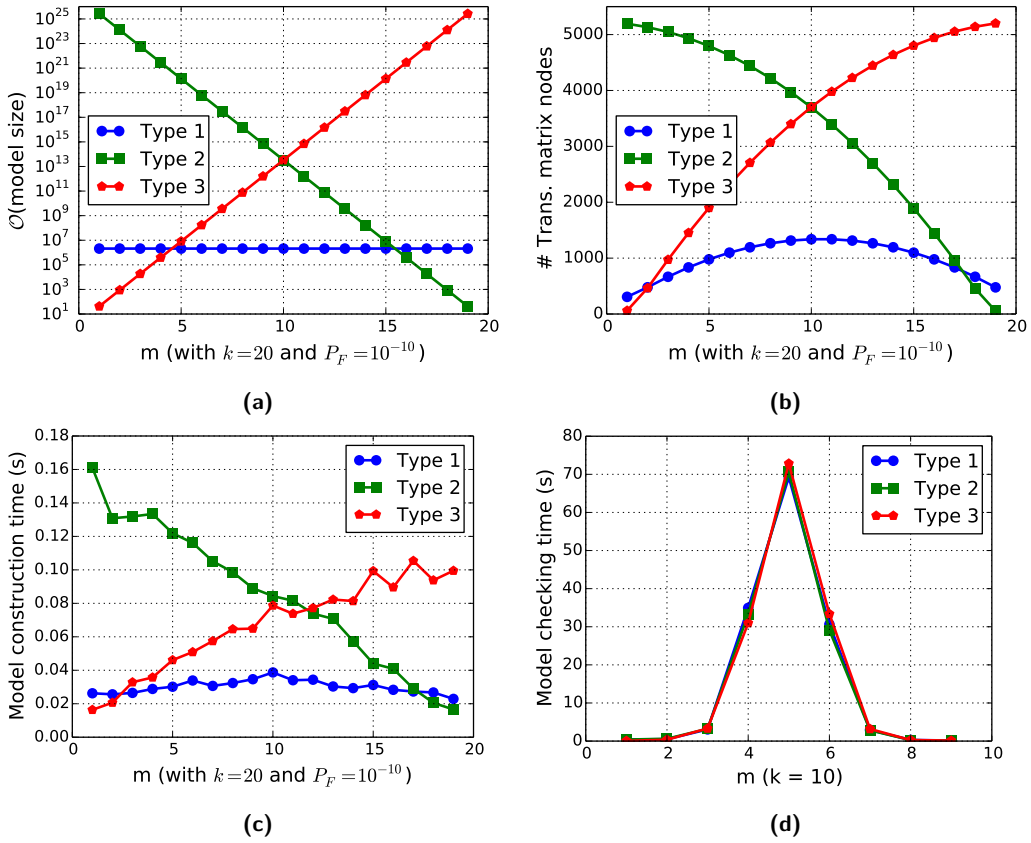


Figure 4 Comparing the three PMC model types using PMC-E. While measuring the checking time statistics, $k = 10$ was used, since model checking for $k = 20$ frequently timed out.

analysis time grows exponentially in k (the y-axis is log scale), PMC's and MRT's analysis times grow super-exponentially. For configurations of type (iii) and (iv) where $m = k - 2$ (Figs. 5c and 5d), PRISM-based analyses scale better than in the first two configurations because the Type-3 model allows for a concise representation of the (m, k) specification in this case and hence fast building of the model. SAP's scalability also improves significantly in this case because the recursion involved in computing $R_{LB}(k - m + 1, k, n - 1)$ for the empirical data points is eliminated in this case. Between configurations (i) and (ii), as well as between configurations (iii) and (iv), only the failure probability P_F is changed from 10^{-10} to 10^{-20}. As a result, PMC-E takes an order of magnitude more time. This is because lower probabilities require more space for exact representation, and hence more time for computations on these representations. SAP is also affected since the number of data points to be measured is larger in this case. MART is unaffected because irrespective of P_F, it uses a precision of 1000. PMC-P is also unaffected since it is independent of P_F.

To summarize the discussion on analysis scalability, we illustrate in Fig. 6a for each $k \in [1, 25]$ and $m \in [2, k - 1]$ whether analyses PMC-P, PMC-E, MART, and SAP finished on time, *i.e.*, within a one-hour timeout window. For each cell, P denotes that PMC-P was successful, E denotes that PMC-P timed out but PMC-E was successful, M denotes that both PMC-P and PMC-E timed out but MART was successful, and S denotes that only SAP was successful. Clearly, the results indicate that exact analyses can be used only if $k \leq 15$, or else if m is either very small or very large relative to k. Thus, for larger values of k, an

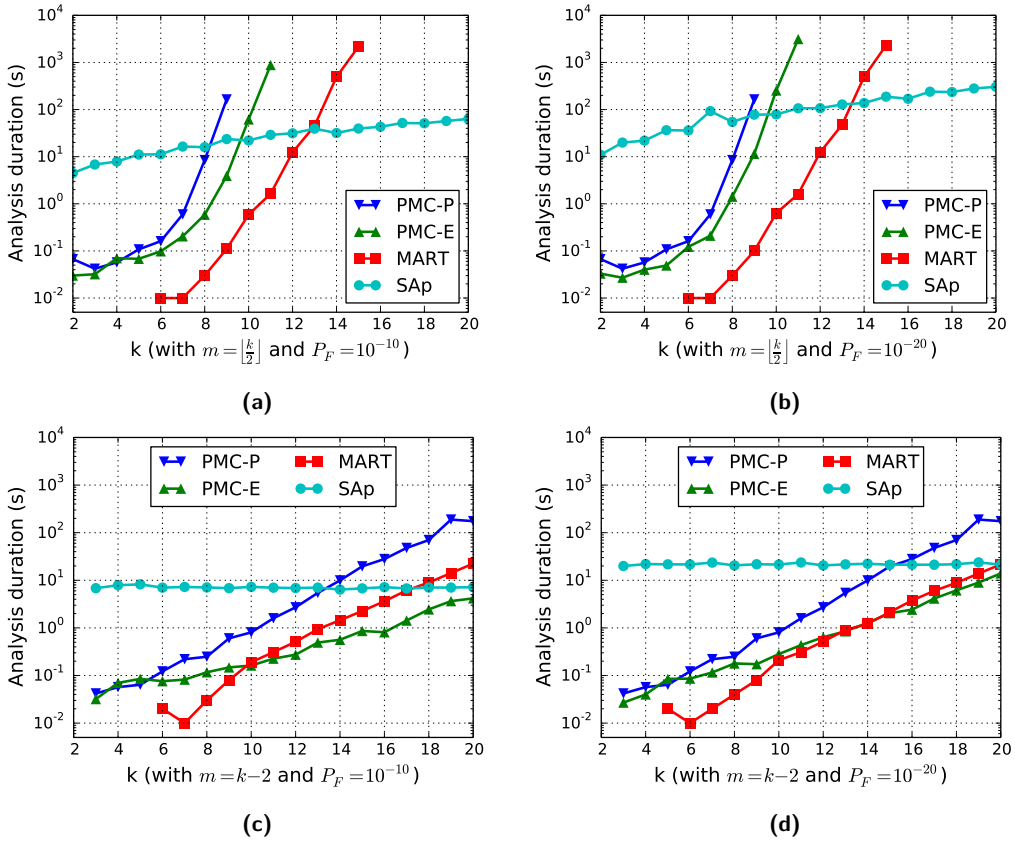
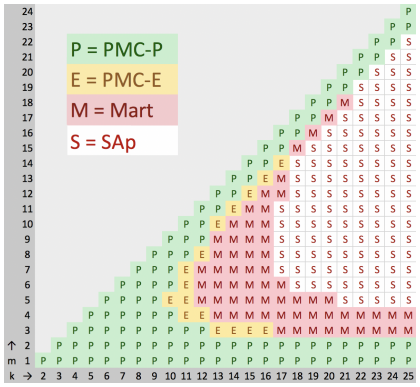


Figure 5 Comparing analysis duration for PMC-P, PMC-E, MART, and SAP. The analysis duration for MART for $k \leq 5$ was extremely small and is hence not illustrated. The configuration $k = 2$ in (c) and (d) was ignored since $(0, 2)$ is not a valid (or rather a trivial) specification.

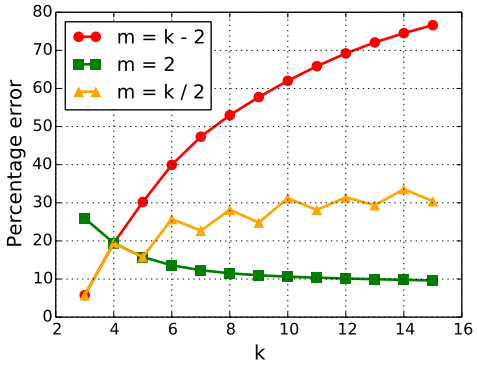
approximate analysis, such as SAP, is needed, that trades some accuracy for scalability. But is SAP accurate enough to be useful at very large values of k? And is it accurate for small values of k so that the costly exact analyses may not be needed at all? To answer these questions, we evaluate next SAP's accuracy with respect to MART and PMC.

In Fig. 6c (similar in structure to Fig. 6a), we report the percentage error in the MTTF obtained using SAP versus that obtained from either PMC or MART (PMC was preferred, if available) for each $k \in [2, 12]$ and $m \in [1, k-1]$. As expected, SAP always resulted in a lower, pessimistic MTTF than PMC and MART since it is sound by construction. Thus, error signs are not explicitly denoted in the figure.

We make the two key observations regarding SAP's accuracy. First, even for small values of k, the relative errors are significant (see the red cells in Fig. 6c denoting specifications with relative error greater than 50%). This validates the need for an exact analysis whenever feasible. Second, the relative errors are higher if the ratio m/k is closer to one. To investigate this further, we also plot the percentage errors for $m = k - 2$, $m = 2$, and $m = k/2$ with respect to k in Fig. 6b. From this figure, we observe that in all evaluated cases, the MTTF estimated with SAP was within an order of magnitude of the exact MTTF. Since in the context of reliability analyses the order of magnitude is typically of prime interest (rather than the exact value), we conclude that SAP is reasonably accurate for large values of k.



(a)



(b)

m \ k	2	3	4	5	6	7	8	9	10	11	12
11											81.45
10										79.25	69.20
9									76.61	65.85	58.25
8								73.38	62.01	54.73	47.79
7							69.36	57.75	50.31	44.30	39.06
6						64.29	52.99	44.91	40.11	35.35	31.43
5					57.84	47.34	39.12	34.28	31.22	28.13	25.06
4				49.62	39.96	33.44	28.17	24.82	22.98	21.64	20.28
3			39.29	30.21	25.73	22.68	20.09	17.80	15.93	14.55	13.63
↑ 2		25.91	19.44	15.77	13.60	12.30	11.50	10.98	10.62	10.35	10.13
m 1	05.76	05.76	05.76	05.76	05.76	05.76	05.76	05.76	05.76	05.76	05.76
k →	2	3	4	5	6	7	8	9	10	11	12

Error > 50 %
25 % < Error ≤ 50 %
Error ≤ 25 %

(c)

Figure 6 Quantifying the scalability vs. accuracy tradeoff. **(a)** Scalability results for different values of m and k. **(b)** SAp's accuracy trend for $m = 2$, $m = k/2$, and $m = k - 2$. **(c)** Summary of SAp's accuracy with respect to Mart and PMC for different values of k and m.

7.4 Discussion

Mart outperforms both PMC-P and PMC-E, which is not surprising. In fact, for the scenario with IID iteration failure probabilities that we evaluated, Mart directly represents the underlying system of linear equations without needing to construct a model. PMC's benefits lie in its ability to express non-IID iteration failure probabilities. SAp on the other hand scales much better than PMC and Mart, at the cost of acceptable, but non-zero pessimism. To conclude, PMC, Mart, and SAp are useful alternatives for reliability evaluation depending on the values of m and k. PMC and Mart are ideal to evaluate *short-range* safety properties that are usually applied on short window lengths *e.g.*, such as "there should not be more than 3 consecutive failures in any window of 10 iterations" [18]. In contrast, SAp can evaluate asymptotic properties that are defined over a large window of events and reflect minimum acceptable longterm *quality-of-service levels*, *e.g.*, such as "at least 90% of actuation commands must be applied on the plant in every 100 iterations" [33, 41].

Although we focused on a binary failure type in this paper, *i.e.*, each iteration was categorized either as a successful iteration or a failed iteration, one could also use fine-grained label types for each iteration, such as deadline violation, message loss, miscomputation, and so on. That is, an execution of system S could be modeled as a string in $\{0 \dots \lambda\}^*$, instead of a string in $\{0, 1\}^*$, where λ is the number of failure categories. Both PMC and Mart easily

extend to such systems. In contrast, SAp has limited extensibility in its current form, since our objective when designing SAp was primarily to scale the evaluation of (m, k) specifications that are widely used in practice. However, the same blueprint could be used to safely approximate other types of robustness specifications as well, *i.e.*, by breaking each specification into smaller events, computing the product of respective event probabilities (or a lower bound), and then reusing Lemma 1 for MTTF estimation. We leave such extensions for future work.

8 Related Work

Weakly-hard constraints have been widely studied in the context of firm real-time systems to represent robustness of a time-sensitive task against occasional timing failures [11, 16, 17, 24, 36, 38]. In particular, the focus has been on **(i)** analyzing task schedulability according to a given weakly-hard (usually (m, k)) constraint [36, 37], **(ii)** design of online schedulers to meet these constraints [11, 16, 17, 24], and **(iii)** co-design approaches to find the schedulable set of (m, k) parameters that maximizes an application's quality of service [18, 29, 45]. Most recently, Pazzaglia *et al.* [34] introduced state-based representation of the evolution of a control system with respect to deadline misses, and showed the merits of having multiple (m, k) constraints for a control application. In contrast, Huang *et al.* [26] focused on the safety verification problem of nonlinear weakly-hard systems by modeling them using hybrid automata. Huang *et al.* [27] also discuss new research directions in applying weakly-hard constraints to general-purpose networked systems. None of these papers provides a means for bounding a system's MTTF with respect to its weakly-hard specification.

In the general reliability literature, there is a long tradition of work on deriving a system's MTTF if the occurrence of failures is described by well-known probability distributions (see [30] for a comprehensive overview). Similarly, the problem of evaluating the reliability of series- or parallel-redundant systems, both with and without repairs, in the context of robustness specifications such as *k-out-of-n*, *consecutive-k-out-of-n*, multidimensional *consecutive-k-out-of-n*, *etc.* is well understood, *e.g.*, see [35, 40]. However, the available techniques in this domain do not directly apply to the problem studied in this paper. Either the constraints cannot be reduced to these techniques or symbolically integrating the applicable technique over an infinite domain is not trivial. Further, for multiple weakly-hard specifications, a model-based approach helps to accurately account for common failure sequences.

9 Conclusion

We proposed methods for safely bounding the MTTF of periodic systems with stochastic faults w.r.t. weakly-hard robustness specifications. Empirical evaluations showed that an exact (even parametric) analysis is feasible when $k - m$ is small, and that an approximate analysis is scalable and (within the parameters of our experiments) produces bounds within $2\times$ of the exact bounds. In future work, it would be interesting to consider more expressive models of stochastic faults [39], such as continuous-time models and dynamic fault trees. We also plan to extend SAp for evaluating other types of robustness constraints. Finally, a holistic analysis of periodic systems that are composed of multiple subsystems with possibly different periods and different weakly-hard constraints, as opposed to analyzing each of these subsystems independently, would help reduce pessimism in the overall reliability assessment.

References

1 BLAS (Basic Linear Algebra Subprograms). URL: http://www.netlib.org/blas/.

2 Elemental: distributed-memory dense and sparse-direct linear algebra and optimization — Elemental. URL: http://libelemental.org/.

3 The GNU MPFR Library. URL: https://www.mpfr.org/.

4 IEC 61158-1:2014 | IEC Webstore. URL: https://webstore.iec.ch/publication/4624.

5 LAPACK – Linear Algebra PACKage. URL: http://www.netlib.org/lapack/.

6 mpmath - Python library for arbitrary-precision floating-point arithmetic. URL: http://mpmath.org/.

7 Open MPI: Open Source High Performance Computing. URL: https://www.open-mpi.org/.

8 Rajeev Alur and Thomas Henzinger. Reactive modules. In *Proceedings 11th Annual IEEE Symposium on Logic in Computer Science*, pages 207–218, New Brunswick, NJ, USA, 1996. IEEE Comput. Soc. Press. URL: http://ieeexplore.ieee.org/document/561320/, doi:10.1109/LICS.1996.561320.

9 Robert B. Ash. *Basic probability theory*. Dover Publications, Mineola, N.Y, dover ed edition, 2008. OCLC: ocn190785258 (pbk.).

10 Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. The MIT Press, Cambridge, Mass, 2008. OCLC: ocn171152628.

11 Guillem Bernat and Alan Burns. Combining (/sub m//sup n/)-hard deadlines and dual priority scheduling. In *Proceedings Real-Time Systems Symposium*, pages 46–57, San Francisco, CA, USA, 1997. IEEE Comput. Soc. URL: http://ieeexplore.ieee.org/document/641268/, doi:10.1109/REAL.1997.641268.

12 Guillem Bernat, Alan Burns, and Albert Liamosi. Weakly hard real-time systems. *IEEE Transactions on Computers*, 50(4):308–321, April 2001. URL: http://ieeexplore.ieee.org/document/919277/, doi:10.1109/12.919277.

13 Rainer Blind and Frank Allgower. Towards Networked Control Systems with guaranteed stability: Using weakly hard real-time constraints to model the loss process. In *2015 54th IEEE Conference on Decision and Control (CDC)*, pages 7510–7515, Osaka, December 2015. IEEE. URL: http://ieeexplore.ieee.org/document/7403405/, doi:10.1109/CDC.2015.7403405.

14 Michael S. Branicky, Stephen M. Phillips, and Wei Zhang. Scheduling and feedback co-design for networked control systems. In *Proceedings of the 41st IEEE Conference on Decision and Control, 2002.*, volume 2, pages 1211–1217, Las Vegas, NV, USA, 2002. IEEE. URL: http://ieeexplore.ieee.org/document/1184679/, doi:10.1109/CDC.2002.1184679.

15 Ian Broster, Alan Burns, and Guillermo Rodriguez-Navas. Timing Analysis of Real-Time Communication Under Electromagnetic Interference. *Real-Time Systems*, 30(1-2):55–81, May 2005. URL: http://link.springer.com/10.1007/s11241-005-0504-z, doi:10.1007/s11241-005-0504-z.

16 Marco Caccamo and Giorgio Buttazzo. Exploiting skips in periodic tasks for enhancing aperiodic responsiveness. In *Proceedings Real-Time Systems Symposium*, pages 330–339, San Francisco, CA, USA, 1997. IEEE Comput. Soc. URL: http://ieeexplore.ieee.org/document/641294/, doi:10.1109/REAL.1997.641294.

17 Hyunjong Choi, Hyoseung Kim, and Qi Zhu. Job-Class-Level Fixed Priority Scheduling of Weakly-Hard Real-Time Systems. In *2019 IEEE Real-Time and Embedded Technology*

and Applications Symposium (RTAS), Montreal, Quebec, Canada, April 2019. IEEE. doi:10.1109/RTAS.2019.00028.

18 Hoon Sung Chwa, Kang G. Shin, and Jinkyu Lee. Closing the Gap Between Stability and Schedulability: A New Task Model for Cyber-Physical Systems. In *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 327–337, Porto, April 2018. IEEE. URL: https://ieeexplore.ieee.org/document/8430094/, doi:10.1109/RTAS.2018.00040.

19 Christian Dehnert, Sebastian Junges, Joost-Pieter Katoen, and Matthias Volk. A Storm is Coming: A Modern Probabilistic Model Checker. In *Computer Aided Verification*, pages 592–600, Cham, 2017. Springer International Publishing.

20 Joanne Bechta Dugan and Randy Van Buren. Reliability evaluation of fly-by-wire computer systems. *Journal of Systems and Software*, 25(1):109–120, April 1994. URL: http://linkinghub.elsevier.com/retrieve/pii/0164121294900612, doi:10.1016/0164-1212(94)90061-2.

21 Oliver Gettings, Sophie Quinton, and Robert I. Davis. Mixed criticality systems with weakly-hard constraints. In *Proceedings of the 23rd International Conference on Real Time and Networks Systems - RTNS '15*, pages 237–246, Lille, France, 2015. ACM Press. URL: http://dl.acm.org/citation.cfm?doid=2834848.2834850, doi:10.1145/2834848.2834850.

22 Arpan Gujarati, Mitra Nasri, and Björn B. Brandenburg. Lower-Bounding the MTTF for Systems with (m, k) Constraints and IID Iteration Failure Probabilities. Technical Report MPI-SWS-2018-004, Max Planck Insitute for Software Systems, April 2018. URL: https://www.mpi-sws.org/tr/2018-004.pdf.

23 Arpan Gujarati, Mitra Nasri, and Björn B. Brandenburg. Quantifying the Resiliency of Fail-Operational Real-Time Networked Control Systems. In *30th Euromicro Conference on Real-Time Systems (ECRTS 2018)*, volume 106 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:24, Barcelona, Spain, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. URL: http://drops.dagstuhl.de/opus/volltexte/2018/8988/, doi:10.4230/lipics.ecrts.2018.16.

24 Moncef Hamdaoui and Parameswaran Ramanathan. A dynamic priority assignment technique for streams with (m, k)-firm deadlines. *IEEE Transactions on Computers*, 44(12):1443–1451, December 1995. URL: http://ieeexplore.ieee.org/document/477249/, doi:10.1109/12.477249.

25 Rafik Henia, Arne Hamann, Marek Jersak, Razvan Racu, Kai Richter, and Rolf Ernst. System level performance analysis – the SymTA/S approach. *IEE Proceedings - Computers and Digital Techniques*, 152(2):148, 2005. URL: https://digital-library.theiet.org/content/journals/10.1049/ip-cdt_20045088, doi:10.1049/ip-cdt:20045088.

26 Chao Huang, Wenchao Li, and Qi Zhu. Formal verification of weakly-hard systems. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems Computation and Control - HSCC '19*, pages 197–207, Montreal, Quebec, Canada, 2019. ACM Press. URL: http://dl.acm.org/citation.cfm?doid=3302504.3311811, doi:10.1145/3302504.3311811.

27 Chao Huang, Kacper Wardega, Wenchao Li, and Qi Zhu. Exploring weakly-hard paradigm for networked systems. In *Proceedings of the Workshop on Design Automation for CPS and IoT - DESTION '19*, pages 51–59, Montreal, Quebec, Canada, 2019. ACM Press. URL: http://dl.acm.org/citation.cfm?doid=3313151.3313165, doi:10.1145/3313151.3313165.

28 Anastasiia Izycheva and Eva Darulova. On Sound Relative Error Bounds for Floating-point Arithmetic. In *Proceedings of the 17th Conference on Formal Methods in Computer-Aided Design*, FMCAD '17, pages 15–22, Austin, TX, 2017. FMCAD Inc. URL: http://dl.acm.org/citation.cfm?id=3168451.3168462.

29 Matthias Kauer, Damoon Soudbakhsh, Dip Goswami, Samarjit Chakraborty, and Anuradha M. Annaswamy. Fault-tolerant Control Synthesis and Verification of Distributed Embedded Systems. In *Proceedings of the Conference on Design, Automation & Test in Europe*, DATE '14, pages 56:1–56:6, 3001 Leuven, Belgium, Belgium, 2014. European Design and Automation Association. URL: http://dl.acm.org/citation.cfm?id=2616606.2616675.

30 Way Kuo and Ming J. Zuo. *Optimal reliability modeling: principles and applications*. John Wiley & Sons, Hoboken, N.J, 2003.

31 Marta Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of Probabilistic Real-Time Systems. In *Computer Aided Verification*, volume 6806, pages 585–591. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. URL: http://link.springer.com/10.1007/978-3-642-22110-1_47, doi:10.1007/978-3-642-22110-1_47.

32 Shuo-Yen Robert Li. A Martingale Approach to the Study of Occurrence of Sequence Patterns in Repeated Experiments. *The Annals of Probability*, 8(6):1171–1176, 1980. URL: https://www.jstor.org/stable/2243018.

33 Rupak Majumdar, Indranil Saha, and Majid Zamani. Performance-aware scheduler synthesis for control systems. In *Proceedings of the 9th ACM international conference on Embedded software - EMSOFT '11*, page 299, Taipei, Taiwan, 2011. ACM Press. URL: http://dl.acm.org/citation.cfm?doid=2038642.2038689, doi:10.1145/2038642.2038689.

34 Paolo Pazzaglia, Luigi Pannocchi, Alessandro Biondi, and Marco Di Natale. Beyond the Weakly Hard Model: Measuring the Performance Cost of Deadline Misses. In *30th Euromicro Conference on Real-Time Systems (ECRTS 2018)*, volume 106 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 10:1–10:22, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. URL: http://drops.dagstuhl.de/opus/volltexte/2018/8993, doi:10.4230/LIPIcs.ECRTS.2018.10.

35 Hoang Pham. Optimal design of k-out-of-n redundant systems. *Microelectronics Reliability*, 32(1):119–126, January 1992. URL: http://www.sciencedirect.com/science/article/pii/002627149290091X, doi:10.1016/0026-2714(92)90091-X.

36 Gang Quan and Xiaobo Hu. Enhanced fixed-priority scheduling with (m,k)-firm guarantee. In *Proceedings 21st IEEE Real-Time Systems Symposium*, pages 79–88, November 2000. doi:10.1109/REAL.2000.895998.

37 Sophie Quinton and Rolf Ernst. Generalized Weakly-Hard Constraints. In *Leveraging Applications of Formal Methods, Verification and Validation. Applications and Case Studies*, Lecture Notes in Computer Science, pages 96–110. Springer Berlin Heidelberg, 2012.

38 Parameswaran Ramanathan. Overload management in real-time control applications using (m, k)-firm guarantee. *IEEE Transactions on Parallel and Distributed Systems*, 10(6):549–559, June 1999. doi:10.1109/71.774906.

39 Enno Ruijters and Mariëlle Stoelinga. Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools. *Computer Science Review*, 15-16:29–62, February 2015. URL: http://www.sciencedirect.com/science/article/pii/S1574013715000027, doi:10.1016/j.cosrev.2015.03.001.

40 Raaj K. Sah. An explicit closed-form formula for profit-maximizing k-out-of-n systems subject to two kinds of failures. *Microelectronics Reliability*, 30(6):1123–1130, January 1990. URL: http://www.sciencedirect.com/science/article/pii/002627149090291T, doi:10.1016/0026-2714(90)90291-T.

41 Indranil Saha, Sanjoy Baruah, and Rupak Majumdar. Dynamic Scheduling for Networked Control Systems. In *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*, HSCC '15, pages 98–107, New York, NY, USA, 2015. ACM. URL: http://doi.acm.org/10.1145/2728606.2728636, doi:10.1145/2728606.2728636.

42 Michael Sfakianakis, Stratis G. Kounias, and Alexander E. Hillaris. Reliability of a consecutive k-out-of-r-from-n:F system. *IEEE Transactions on Reliability*, 41(3):442–447, September 1992. doi:10.1109/24.159817.

43 Purnendu Sinha. Architectural design and reliability analysis of a fail-operational brake-by-wire system from ISO 26262 perspectives. *Reliability Engineering & System Safety*, 96(10):1349–1359, October 2011. URL: http://linkinghub.elsevier.com/retrieve/pii/S095183201100041X, doi:10.1016/j.ress.2011.03.013.

44 Fedor Smirnov, Michael Glaß, Felix Reimann, and Jürgen Teich. Formal reliability analysis of switched ethernet automotive networks under transient transmission errors. In *Proceedings of the 53rd Annual Design Automation Conference on - DAC '16*, pages 1–6, Austin, Texas, 2016. ACM Press. URL: http://dl.acm.org/citation.cfm?doid=2897937.2898026, doi:10.1145/2897937.2898026.

45 Damoon Soudbakhsh, Linh T. X. Phan, Oleg Sokolsky, Insup Lee, and Anuradha Annaswamy. Co-design of Control and Platform with Dropped Signals. In *Proceedings of the ACM/IEEE 4th International Conference on Cyber-Physical Systems*, ICCPS '13, pages 129–140, New York, NY, USA, 2013. ACM. URL: http://doi.acm.org/10.1145/2502524.2502542, doi:10.1145/2502524.2502542.

46 Susan Stanley. MTBF, MTTR, MTTF & FIT Explanation of Terms. URL: http://www.bb-elec.com/Learning-Center/All-White-Papers/Fiber/MTBF,-MTTR,-MTTF,-FIT-Explanation-of-Terms/MTBF-MTTR-MTTF-FIT-10262012-pdf.pdf.

10 Appendix: Implementing PMC in PRISM

In §4, we introduced PMC, an approach based on Markov chain analysis to estimate the exact MTTF of periodic systems with weakly-hard constraints. As per PMC, the system is modeled as a labeled discrete-time Markov chain M. Given a robustness specification of $\mathcal{R} = (m, k)$, another monitor Markov chain $Monitor(M, \mathcal{R})$ (classified as a Type 1 monitor) runs alongside M. Each step of the monitor is assumed to have a reward of 1. The set of all states in $Monitor(M, k)$ that violate the (m, k) constraints are denoted $Bad(m, k)$ and are made *absorbing*. The MTTF of the system is then given by $T \times E$, where T denotes the system period, and E denotes the expected reward until any state in $Bad(m, k)$ is reached starting from an initial state. In this appendix, we explain how the Type-1 monitor representation $Monitor(M, k)$ and its optimized variants (Type 2 and Type 3) can be encoded in the PRISM language, and given an encoded model, how the expected reward E is computed.

In the end, we report on comparison of PRISM's performance with that of STORM, which is a more recent probabilistic model checker.

10.1 Example

PRISM accepts discrete-time Markov chains described using a state-based modeling language based on the *reactive modules* formalism of Alur and Henzinger [8]. For an example robustness specification of $(5, 10)$, we illustrate implementations of the corresponding Type-1, Type-2, and Type-3 monitors in the PRISM language in Listings 1, 2, and 3, respectively. These implementations are explained in brief in the following.

10.1.1 Type-1 Monitor

The keyword `dtmc` indicates to PRISM that the following model should be interpreted as a *discrete time Markov chain*. Constant `q` denotes the iteration failure probability bound P_F. Each boolean variable `si` keeps track of whether the i^{th} most recent iteration of the system was successful (`true`) or not (`false`). The formula `num_failures_k_1` thus computes the number of failed iterations among the last $k-1$ consecutive iterations. It is used to decide whether a new iteration results in the violation of the (m, k) robustness specification. That is, if the new iteration is not successful, and if the last $k-1$ consecutive iterations already consisted of $k-m$ or more failures (`num_failures_k_1` $\geq k-m$), less than m iterations are successful in the last k consecutive iterations and hence the (m, k) specification is violated. Alternatively, if `num_failures_k_1` $< k-m$ (defined using formula `failure_allowed`), even if the new iteration is not successful, the (m, k) specification is not violated.

A single step of the monitor corresponds to an execution of one iteration of the system. The command in Lines 29–36 updates the global state at the end of each step. Note that `si'` denotes the updated state of variable `si`. Since the i^{th} latest iteration *before* the step corresponds to the $i+1^{\text{st}}$ latest iteration *after* the step, for each $2 \leq j = i+1 \leq k$, variable `sj` is updated to `si` (*e.g.*, `s4'=s3`).

If the latest iteration is successful, which happens with probability `p`, variable `s1` is updated to `true`; otherwise, with probability `q`, variable `s1` is updated to `false`. In the latter case, the command also check if the (m, k) specification is violated, and updates the `safe` variable accordingly. Note that variable `safe` is mainly used to simplify the property specification (described below). The *reward* structure at the end of the listing (Lines 40–42) associates a reward of one with each step, which is then used to compute the MTTF.

Listing 1 Type-1 PRISM model for $(5, 10)$

```
dtmc

const int m = 5;
const int k = 10;
const double q = 1e-10;
formula p = 1.0 - q;

formula num_failures_k_1 = (s1?0:1) + (s2?0:1) + (s3?0:1) +
                           (s4?0:1) + (s5?0:1) + (s6?0:1) +
                           (s7?0:1) + (s8?0:1) + (s9?0:1);

formula failure_allowed = (num_failures_k_1 < k-m);

module reliability_analysis

s1  : bool init true;
s2  : bool init true;
s3  : bool init true;
s4  : bool init true;
s5  : bool init true;
s6  : bool init true;
s7  : bool init true;
s8  : bool init true;
s9  : bool init true;
s10 : bool init true;

safe : bool init true;

[] true -> p: (safe'=safe)
              & (s1'=true)  & (s2'=s1) & (s3'=s2) & (s4'=s3)
              & (s5'=s4)    & (s6'=s5) & (s7'=s6) & (s8'=s7)
              & (s9'=s8)    & (s10'=s9)
         + q: (safe'=safe & failure_allowed)
              & (s1'=false) & (s2'=s1) & (s3'=s2) & (s4'=s3)
              & (s5'=s4)    & (s6'=s5) & (s7'=s6) & (s8'=s7)
              & (s9'=s8)    & (s10'=s9);

endmodule

rewards "steps"
    true : 1;
endrewards
```

Listing 2 Type-2 PRISM model for $(5, 10)$

```
dtmc

const int m = 5;
const int k = 10;
const double q = 1e-10;
formula p = 1.0 - q;

formula failure_allowed = ((s1<2)|(s2<2)|(s3<2)|(s4<2)|(s5<2));

module reliability_analysis

s1 : [0..k] init 0;
s2 : [0..k] init 0;
s3 : [0..k] init 0;
s4 : [0..k] init 0;
s5 : [0..k] init 0;

safe : bool init true;

[] true -> p: (safe'=safe)
               & (s1'=((s1>0)?(s1-1):0)) & (s2'=((s2>0)?(s2-1):0))
               & (s3'=((s3>0)?(s3-1):0)) & (s4'=((s4>0)?(s4-1):0))
               & (s5'=((s5>0)?(s5-1):0))
         + q: (safe'=safe & failure_allowed)
               & (s1'=k)                  & (s2'=((s1>0)?(s1-1):0))
               & (s3'=((s2>0)?(s2-1):0)) & (s4'=((s3>0)?(s3-1):0))
               & (s5'=((s4>0)?(s4-1):0));

endmodule

rewards "steps"
        true : 1;
endrewards
```

Once the model file is built, it can be queried with temporal logic queries, which must be specified in PRISM's property specification language. In particular, to compute the MTTF, we query the model with the reward-based property `R=? [ F safe=false ]`, which is essentially asking PRISM the following question: "what is the expected reward accumulated (denoted `R`) until (*i.e.*, operator `F`) the safety property is violated (`safe=false`)?" To answer the question, PRISM's engine performs a reachability analysis over the model's state space. Since we associate one reward per step, PRISM in this case returns the expected number of steps to failure, and the result can then be multiplied with the system's time period T to obtain the MTTF. Alternatively, one could simply associate a reward of T with each step.

10.1.2 Type-2 Monitor

Listing 2 illustrates the optimized monitor representation of Type 2. In this case, only $k - m = 5$ variables (`s1`-`s5`), instead of $k = 10$ variables, are needed to capture the global state, *i.e.*, the status of last k consecutive iterations. However, each variable si denotes the

Listing 3 Type-3 PRISM model for $(5, 10)$

```
1  dtmc
2
3  const int m = 5;
4  const int k = 10;
5  const double q = 1e-10;
6  formula p = 1.0 - q;
7
8  formula failure_allowed = ((s1>1)&(s2>1)&(s3>1)&(s4>1)&(s5>1));
9
10 module reliability_analysis
11
12 s1 : [0..k] init (k-0);
13 s2 : [0..k] init (k-1);
14 s3 : [0..k] init (k-2);
15 s4 : [0..k] init (k-3);
16 s5 : [0..k] init (k-4);
17
18 safe : bool init true;
19
20 [] true -> p: (safe'=safe)
21               & (s1'=k)
22               & (s2'=((s1>1)?(s1-1):0)) & (s3'=((s2>1)?(s2-1):0))
23               & (s4'=((s3>1)?(s3-1):0)) & (s5'=((s4>1)?(s4-1):0))
24          + q: (safe'=safe & failure_allowed)
25               & (s1'=((s1>1)?(s1-1):0)) & (s2'=((s2>1)?(s2-1):0))
26               & (s3'=((s3>1)?(s3-1):0)) & (s4'=((s4>1)?(s4-1):0))
27               & (s5'=((s5>1)?(s5-1):0));
28
29 endmodule
30
31 rewards "steps"
32         true : 1;
33 endrewards
```

position of a failed iteration (among the last k consecutive iterations) and thus takes up to $k+1$ different values (0 is a sentinel value that indicates that variable `si` does not point to a failed iteration). In addition, the global state update (Lines 20–27) ensure that each variable `si` points to a unique failed iteration (*i.e.*, $\nexists i, j \ : \ 1 \leq i, j \leq k - m \ \wedge \ \texttt{si} = \texttt{sj} \neq 0$). In particular, variable `si` always points to the i^{th} most recent failed iteration.

Since (m, k) robustness allows for up to $k - m$ failed iterations, a new failed iteration violates safety only if there are already $k-m$ failed iterations among the last $k-1$ consecutive iterations (*i.e.*, for each $1 \leq i \leq k - m$, variables $\texttt{si} \geq 2$). The status of the (k^{th}) oldest iteration does not matter after the new iteration is executed, since it does not affect the (m, k) robustness specification anymore. In other words, the system remains safe despite the new iteration being not successful if for some $1 \leq i \leq k-m$, $\texttt{si} < 2$ (*i.e.*, `failure_allowed`).

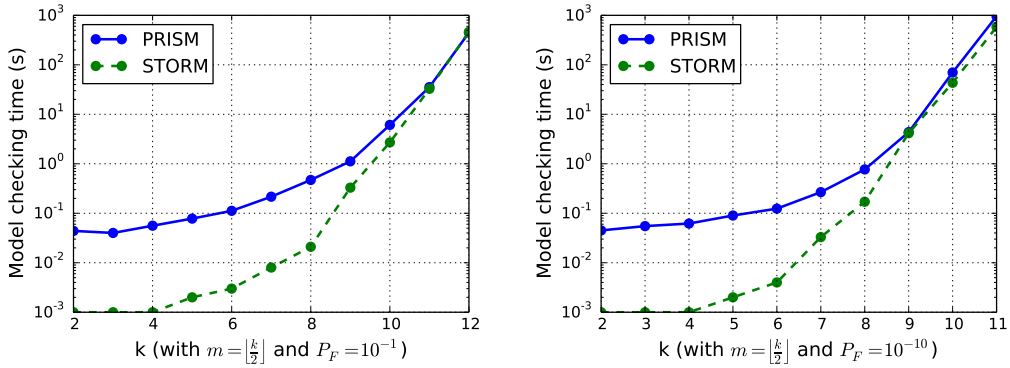


Figure 7 Exact model checking in PRISM and STORM.

10.1.3 Type-3 Monitor

Listing 3 illustrates the Type-3 monitor representation. It is similar to the Type-2 representation, except that state variables in this case track the successful iterations instead of the failed iterations (since Type 2 is optimized for $m \ll k$ and not $k - m \ll k$). In particular, the representation uses $m = 5$ variables (`s1-s5`), where each variable `si` denotes the position of the i^{th} most recent successful iteration. A failed iteration does not violate the (m, k) specification as long as there are at least m successful iterations among the last $k - 1$ consecutive iterations, *i.e.*, for each $1 \leq i \leq m$, variable `si` > 1 (denoted as formula `failure_allowed` in the listing). Once again, the status of the oldest iteration does not matter after the new iteration is executed, since it does not affect (m, k) robustness anymore.

10.2 Comparisons with Storm

While PRISM is a state-of-the-art probabilistic model checker and has been widely used for probabilistic analyses, STORM provides another, more recent alternative for MTTF estimation. In fact, Dehnert *et al.* [19] reported that for *exact* model checking (which is needed for numerical precision), STORM performs better than PRISM by up to three orders of magnitude. However, when we compared the performance of the two model checkers in the context of our MTTF estimation problem, we observed that both tools have similar performance for $k > 10$ and $k > 8$ when $P_F = 10^{-1}$ and $P_F = 10^{-10}$, respectively (the results are illustrated in Figure 7). Hence, we favored PRISM over STORM owing to its better tool support.